LESSON 10: PASSING VARIABLES IN A URL

WEB SYSTEMS AND TECHNOLOGIES 2

OBJECTIVES

1. Understand how to pass variables in a Uniform Resource Locator.
2. Learn how to retrieved variables being passed.

When you work with ASP, you often need to pass variables from one page to another. This lesson is about passing variables in a URL.

HOW DOES IT WORK?

Maybe you have wondered why some URLs looks something like this:

```
http://html.net/page.asp?id=1254
```

Why is there a question mark after the page name?

The answer is that the characters after the question mark are a **HTTP query string**. A HTTP query string can contain both variables and their values. In the example above, the HTTP query string contains a variable named id, with the value 1254.

Here is another example:

```
http://html.net/page.asp?navn=Joe
```

Again, you have a variable (name) with a value (Joe).

HOW TO GET THE VARIABLE WITH ASP?

Let's say you have an ASP page named **people.asp.** Now you call this page using the following URL:

```
people.asp?name=Joe
```

With ASP you will be able to get the value of variable 'name' like this:

```
Request.QueryString("name")
```

So, you use the object **Request** and **QueryString** to find the value of a named variable. Let's try it in an example:

```
<html>
<head>
<title>QueryString</title>
</head>
<body>
<%

' The value of the variable is found

Response.Write "<h1>Hello " & Request.QueryString("name") & "</h1>"
%>
</body>

</html>
```

When you look at the example above, try to replace the name "Joe" with your own name in the URL and then call the document again! Quite nice, eh?

SEVERAL VARIABLES IN THE SAME URL

You are not limited to pass only one variable in a URL. By separating the variables with **&**, multiple variables can be passed:

```
people.asp?name=Joe&age=24
```

This URL contains two variables: name and age. In the same way as above, you can get the variables like this:

```
Request.QueryString("name")
Request.QueryString("age")
```

Let's add the extra variable to the example:

```
<html>
<head>
<title>QueryString </title>
</head>
<body>
<%
' The value of the variable name is found

Response.Write "<h1>Hello" & Request.QueryString("name") & "</h1>"

' The value of the variable age is found

Response.Write "<h1>You are " & Request.QueryString("age") & " years old</h1>"

%>

</body>
</html>
```

WHAT'S NEXT

Now you have learned one way to pass values between pages by using the URL. In the next lesson, we'll look at another method: forms.

REFERENCE

1. HTML Forms - https://www.w3schools.com/html/html_forms.asp
2. ASP forms and User Input - https://www.w3schools.com/asp/asp_inputforms.asp

LESSON 11: PASSING VARIABLES WITH FORMS

OBJECTIVES

1. Understand the principle in using variables with form *request.form* method
2. Learn how to retrieved and handled variables in a form.

PASSING VARIABLES WITH FORMS

Interactive websites require input from users. One of the most common ways to get input are with forms.

In this lesson, we will look at how to build forms and process inputs on the server.

<form>
When you code a form, there are two particular important attributes: **action** and **method**.

Action
Is used to enter the URL where the form is submitted. It this case it would be the ASP file that you want to handle the input.

Method
Can either have the value "post" or "get" which are two different methods to pass data. At this point, you don't need to know much about the difference, but with "get", the data is sent through the URL, and with "post", data is sent as a block of data through standard input service (STDIN).

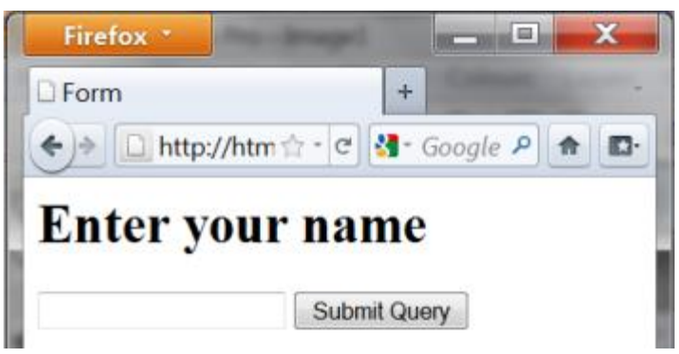
AN HTML PAGE WITH A FORM

The page that contains the form doesn't need to be an ASP file (but it can be). It need not even be on the same site as the file that will receive the data.

In our first example, we will look at a very simple form with one text field:

```
<html>
<head>
<title>Form</title>
</head>
<body>

<h1>Enter your name</h1>

<form method="post" action="handler.asp">
<input type="text" name="username">
<input type="submit">
</form>

</body>
</html>
```

The result in the browser is a form:



Now we come to the fun part: receiving and handling the data with ASP.

REQUESTING FORM DATA WITH ASP

When you need to request data from a form - and many other places (we will come back to that) - you use the object **request**. Since our data was submitted through a form using the method "post", we write:

```
Request.Form("fieldname")
```

Which returns the value of the text field in the form. Let us try to use it in an example.

First create a page with a form as above. Then make an ASP page named "handler.asp" (notice that this is the name of the page we wrote in the action attribute in our <form>).

The file "handler.asp" shall have the following content:

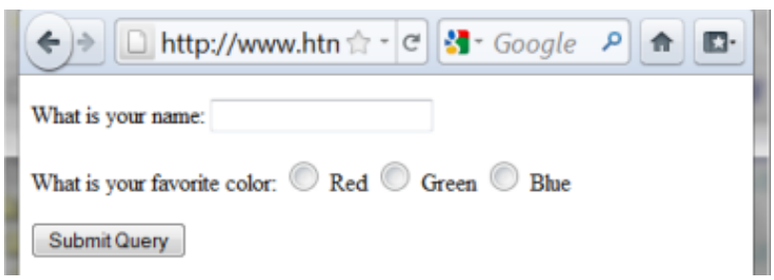
```
<html>
<head>
<title>Form</title>
</head>

<body>
<%
Response.Write "<h1>Hello " & Request.Form("username") & "</h1>"

%>
</body>

</html>
```

USER INPUT AND CONDITIONS

In the next example, we will try to use user input to create conditions. First, we need a form:

```
<html>
<head>
<title>Form</title>
</head>
<body>

<form method="post" action="handler.asp">

<p>What is your name:</p>

<input type="text" name="username"></p>

<p>What is your favorite color:
<input type="radio" name="favoritecolor" value="r" /> Red

<input type="radio" name="favoritecolor" value="g" /> Green
<input type="radio" name="favoritecolor" value="b" /> Blue </p>

<input type="submit" value="Submit" />

</form>

</body>
</html>
```

Which will look like this in the browser:



Now we will use these inputs to create a page that automatically changes background color according to what the user's favorite color is. We can do this by creating a condition *(see lesson 6*) that uses the data the user has filled out in the form.

```
<%
strHeading = "<h1>Hello " & Request.Form("username") & "</h1>"

Select Case Request.Form ("favoritecolor")
Case "r"
       strBackgroundColor = "rgb(255,0,0)"
Case "g"
       strBackgroundColor = "rgb(0255.0)"

Case "b"
       strBackgroundColor = "rgb(0,0,255)"

Case Else
       strBackgroundColor = "rgb(255,255,255)"
End Select
%>

<html>
<head>
<title>Form</title>

</head>
<body style="background: <% = strBackgroundColor %>;">

<% Response.Write strHeading %>

</body>
</html>
```

The background color will be white if the user has not chosen any favorite color in the form. This is done by using **Case Else** to specify what should happen if none of the above conditions are met.

But what if the user does not fill out his name? Then it only says "Hello" in the title. We will use an extra condition to change that.

```
<%
strUserName = Request.Form ("username")

If strUserName <>"" Then

    strHeading = "<h1>Hello " & strUserName & "</h1>"

Else
    strHeading = "<h1>Hello stranger!</h1>"

End If


Select Case Request.Form ("favoritecolor")

Case "r"
       strBackgroundColor = "rgb(255,0,0)"
Case "g"
       strBackgroundColor = "rgb(0255.0)"
Case "b"

       strBackgroundColor = "rgb(0,0,255)"
Case Else
       strBackgroundColor = "rgb(255,255,255)"
End Select
%>

<html>

<head>

<title>Form</title>
```

```
        </head>
        <body style="background: <% = strBackgroundColor %>;">

        <% Response.Write strHeading %>

        </body>
        </html>
```

In the example above we use a condition to **validate** the information from the user. In this case, it might not be so important if the user did not write his name. But as you code more and more advanced stuff, it's vital that you take into account that the user may not always fill out forms the way you had imagined.

REFERENCE

1. HTML Forms - https://www.w3schools.com/html/html_forms.asp
2. ASP forms and User Input - https://www.w3schools.com/asp/asp_inputforms.asp

FOR SUBMISSION

1. Create a script that will collect and display the values and the variable used (at least three variables and values) in the page and in the browser address bar using Request QueryString. Use form elements to pass the values. (upload to webhost under a folder **RQS**)

2. Create a script that will collect and display the values and the variable used (at least three variables and values) in the page and in the browser address bar using Request Form. Use form elements to pass the values. (upload to webhost under a folder **RQF**)

3. Differentiate the used to Request QueryString versus Request.Form. (upload to classroom)

LESSON 12: SESSIONS

HEADING 1

1. Learned how to store and retrieve information about a user from visit to visit.
2. Demonstrate how to retrieve and display information in the page.

SESSIONS

When you visit a website, you do a number of different things. You click from one page to another. Perhaps you also fill out a form or purchase a product.

As a web developer, such information is of great importance to developing successful web solutions.

Suppose, for example, that you want to make a site where some pages are protected with login and password. To make this protection effective, the password-protected pages should have access to information on whether the user has logged in at an earlier time. You must, in other words, be able to "remember" what the user did earlier.

This is exactly what this lesson is about - how you can use Sessions in ASP to store and retrieve information during a user's visit to your site.

SESSION OBJECT

The **Session object** allows you to manage information about a user's session. You can write smart applications that can identify and gather information about users.

A session can begin in different ways. We will not go into technical details here but focus on the case where a session starts by a value is being stored in the Session object. A session ends/dies if the user hasn't requested any pages within in a certain timeframe (by the standard 20 minutes). Of course, you can also always end/kill a session in your script.

Let us say, 50 people are clicking around on the same site, eg. a web shop, at the same time. Information on what they each of them have in their shopping cart would best be stored in the Session object. In order to identify the individual users the server uses a unique user ID that is stored in a cookie. A cookie is a small text file stored on the user's computer - more about cookies in **lesson 13**. Therefore, sessions often require support of cookies in the user's browser.

AN EXAMPLE OF USING SESSIONS

When you requested this page, I stored the current time in a session. I did this so that I can now show you an example of how a session works.

I named the item "StartTime" and stored it by adding the following line in my ASP script:

```
<%
Session ("StartTime") = Now
%>
```

Thereby, a session was started. As described above, each session is given an ID by the server.

Your session has the following ID: **k7v6e7eiet4se6a01f3ckd2kr2**

At any time, I can call the "StartTime" from the session by writing:

```
<%
Response.Write Session("StartTime")
%>
```

Which would reveal that the page was requested at 2/26/2021 6:54:55 (according to the clock on this web server).

But what is interesting is that the information remains in the Session object, even after you have left this page. The information will follow you until your session ends.

By default, a session lasts for 20 minutes, then it dies automatically. But if you want a session to last longer or shorter, you can define the desired time in minutes this way:

```
<%
Session.Timeout = 60
%>
```

In this case, the session will last for 60 minutes before it dies. Too many sessions at the same time overload the server. Therefore, you should not let sessions run longer than necessary.

If you want to stop a session, it can always be killed in this way:

```
<%
Session.Abandon
%>
```

Let us try to look at another example where sessions are used: a password solution.

LOGIN SYSTEM WITH SESSIONS

In the following example, we will make a very simple login system. We will use many of the things they have learned in previous lessons.

The first thing we need is a form where people can enter their username and password. It could look like this:

```
<html>
<head>
<title>Login</title>

</head>
<body>
<form method="post" action="login.asp">

<p>Username: <input type="text" name="username" /></p>
<p>Password: <input type="text" name="password" /></p>

<p><input type="submit" value="Let me in" /></p>

</form>
</body>
</html>
```

Then we create the file login.asp.

In this file, we check whether it is the correct username and password that has been entered. If that is the case, we set a session that says that this user is logged in with the correct username and password.

```
<html>

<head>
<title>Login</title>

</head>
<body>
<%
' Check if username and password are correct
If Request.Form("username") = "asp" AND Request.Form("password") = "asp" Then

' If correct, we set the session to YES
  Session("login") = "YES"
  Session.Timeout = 30
  Response.Write "<h1> You are now logged in</h1>"

  Response.Write "<p><a href='document.asp'>Link to protected file</a> </p>"

Else
```

```
'If not correct, we set the session to NO

  Session("login") = "NO"
  Session.Timeout = 30
  Response.Write "<h1>You are NOT logged in</h1>"

  Response.Write "<p> <a href='document.asp'>Link to protected file</a></p>"

End If
%>

</body>
</html>
```

In the protected files, we want to check whether the user is logged in properly. If this is not the case, the user is sent back to the login form. This is how the protection is made:

```
<%
' If the user is not logged in
' send him/her to the login form

If Session ("login") <>"YES" Then

  Response.Redirect "form.asp"
End If
%>

<html>
<head>
<title>Login</title>
</head>

<body>
<h1>This document is protected</h1>

<p>You can only see it if you are logged in.</p>
</body>
</html>
```

Now you've been introduced to the Session object. In the next lesson we are in same alley and take a closer look at cookies.

REFERENCES

1. ASP Session Object - https://www.w3schools.com/asp/asp_ref_session.asp

SUBMISSION

1. Perform the LOGIN SYSTEM WITH SESSION
2. In your webhost, create a folder (LOGIN SYSTEM) and upload all required files. Noted: No extra files necessary.